

Структуры и классы

- `class`
 - по-умолчанию все поля, методы приватные
 - нечто нетривиальное, требующее инвариантов полей.
- `struct`
 - по-умолчанию все поля, методы публичные
 - обычно что-то простое, не требующее инвариантов полей.

Классы

```
class Rectangle {  
    using SizeType = uint32_t;  
    using AttributesType = uint64_t;  
  
public:  
    Rectangle(SizeType width, SizeType height);  
  
    AttributesType Area() const;  
  
    AttributesType Perimeter() const;  
  
    void SetWidth(SizeType width);  
  
    void SetHeight(SizeType height);  
  
private:  
    uint32_t width_;  
    uint32_t height_;  
}
```

Mutable

Квалификатор для определения поля, как мутабельного. То есть даже в `const` методах данное поле можно будет менять.

Static

```
static int Foo() {
    static int static_variable a = 1;
    return a;
}

class Boo {
public:
    static int Func() {
        static int static_variable = 1;
        return static_variable;
        // return field_;
        // compilation error!
    }

private:
    int field_{1};
};

int main() {
    std::cout << Foo::static_variable << ' '
        << Foo() << ' '
        << Boo::Func() << ' '
        << Boo::Func::static_variable << std::endl;
}
```

Explicit

```
class User {
public:
    User(const std::string& nickname)
        : nickname_(nickname) {

    }

    const std::string& Nickname() const {
        return nickname_;
    }

private:
    std::string nickname_;
};

void PrintNickname(User user) {
    std::cout << user.Nickname() << std::endl;
}

int main() {
    User user = "hello";
    PrintNickname(user);
    PrintNickname("world");
}
```

Explicit

```
class User {
public:
    explicit User(const std::string& nickname)
        : nickname_(nickname) {

    }

    const std::string& Nickname() const {
        return nickname_;
    }

private:
    std::string nickname_;
};

void PrintNickname(User user) {
    std::cout << user.Nickname() << std::endl;
}

int main() {
    User user("hello");
    PrintNickname(user);
    PrintNickname(User("hello"));
}
```

Выделение динамической памяти

```
auto* ptr_int = new int{123};  
auto* ptr_array = new int[123];  
delete[] ptr_array;  
delete ptr_int;
```

Smart pointers

- `std::unique_ptr`
- `std::shared_ptr`